The Report committee for Khurram Shahzad

Certifies that this is the approved version of the following report

Functional Test Automation Framework for Domain Experts

Approved by

Supervising Committee:

Supervisor: _____

Sarfraz Khurshid

_____

William Bard

Functional Test Automation Framework for Domain Experts

by

Khurram Shahzad, B.S.E.E.; M.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2013

Dedication

To my family

Abstract

Functional Test Automation Framework for Domain Experts

by

Khurram Shahzad, M.S.E.

The University of Texas at Austin, 2013

Supervisor: Sarfraz Khurshid

Functional Test of any given system is used to verify that the top level system is performing according to the specifications and all of the sub systems, i.e., hardware, software, inputs, outputs and sensors are operating properly. The term *System* is used here in context of any device or equipment consisting of hardware, software, sensors, virtual sensors and inputs / outputs. One of the examples of such a system is a semiconductor fabrication equipment. There have only been a few approaches that are used to perform the functional test of a system. Engineers typically develop custom test equipment to interface with the system under test and determine proper functioning of all the sub systems and behavior of overall system. In another method, domain experts, utilize the software of the system under test, and manually actuate / stimulate sub systems and then observe and record outcomes to determine whether the system exhibited correct behavior.

A novel solution of a reusable functional test automation framework is developed. The framework interfaces with the system under test via the exposed API, and allow domain experts with minimal or no programming background to create test suites to methodically test overall functionality of the system.

Table of Contents

List of Tables

List of Figures

Functional Test Automation Framework for Domain Experts

Abstract

This project addresses the difficulty involved with the functional test of any medium to complex system (here the term system is used in the context of any device or equipment consisting of hardware, software, sensors, virtual sensors and inputs / outputs). One of the examples of such a system is a semiconductor fabrication equipment. A typical semiconductor fabrication tool has hundreds of sensors, electro-mechanical assemblies, and tens of sub systems. Functional test is used to verify, that the top level system is performing according to the specifications and all of the sub systems, i.e., hardware, software, I/O and sensors are operating properly. There have only been a few conventional approaches that are employed to perform functional test of any system. Following is a brief explanation of traditional system level test methods.

1) Develop custom test equipment which interfaces with the system under test and determine proper functioning of all the sub systems and behavior of overall system.

2) Domain experts utilize the software of the system under test, manually actuate / stimulate sub systems, and then observe and record outcomes to determine whether the system exhibited correct behavior or else troubleshoot, fix, and repeat the above steps again, until satisfactory results are achieved.

Both of the above methods are fairly expansive. Custom test equipment requires hardware design, hardware fabrication, software development, and hardware software integration. In addition to the hefty capital cost involved with the hardware there is also a long

cycle-time associated with the development of custom software. The second approach discussed above is very manual, not repeatable, prone to human mistakes and requires in depth domain expertise to perform full system level functional test. Due to the manual nature of the tests, record keeping is cumbersome and challenging. To mitigate these difficulties a novel solution of functional test automation framework for domain experts is developed. Functional test framework interfaces with the system under test via the API exposed by the system software, and carry out actions, read sensors, and record results. Domain experts with minimal or no prior programming skills can create test scripts in this framework's environment to methodically test subsystems and system level functionality.

Keywords: SUT System under test, the term System is referred to as any intelligent device, equipment or electro-mechanical assembly which has hardware, software, sensors and or inputs/outputs. Embedded Test capability of any System with inherent testing capabilities. **DAQ** data-acquisition. **ROI** return on investment, **COTS** commercial off the shelf, **QTP** quality test professional functional test framework from Hewlet Packard.

Introduction

It is an age old question whether to develop a functional test automation application in house or to get something closest to what is needed off the shelf. Time and again test automation efforts fail due to misjudging the effort and resources necessary to implement a framework verses what is out there available from the COTS world. Most often organizations look outward to get an off the shelf solution to save development time. Test automation vendors usually portray very harmonious inter-working relationship between their test automation framework and the SUT. However, after all the marketing gimmicks and sale pitches when the framework is acquired and put up against the real application all of the inefficiencies and incompatibilities start to show up, and you wonder why this was not apparent during the buying process [4].

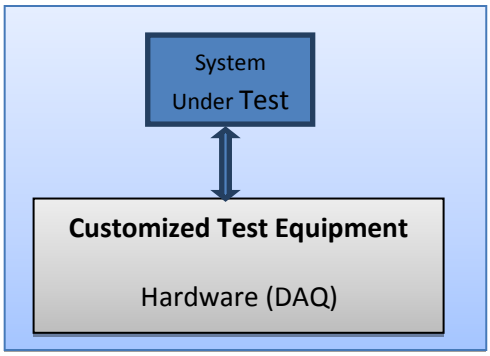


Figure-1 Custom Test Equipment

In this project an automation framework is developed for domain experts to perform functional tests on a fully integrated system. System level test is carried out when all of the sub-modules are integrated together, at that point there is a need for a comprehensive test at a system level to verify that working correctly, and there are no functional, performance or handshaking problems. System level test ensures that the final product meets or exceeds customer requirements. Traditional System level test is carried out by creating custom test equipment. Custom test equipment is always a costly venture and its development lifecycle involves requirements acquisition, hardware / software designs, implementation, qualify and debug (Fig-1). Functional or a system level test provides the ability to verify whether the overall system is doing what it is supposed to do [1], and it provides test coverage for both hardware and software. There is no

question, that comprehensive functional test is critical for any product's success. Functional test is mainly what the end users care about. Functional test capture user requirements in a unique way, and give users and developers confidence that system meet those requirements. Prior to the development of the framework let's look at some of the key requirements for such a framework from domain experts.

1) The framework should be simple and easy to use. The users of the framework will be domain experts and not programmers. Framework users will have very limited or absolutely no knowledge of software development.
2) The framework should be highly reusable. What this means is that the design of the test automation framework should be flexible enough such that the interface between framework and different domains can be accomplished relatively with ease and very minimal effort. This feature will immensely improve ROI (return on investment).
3) The system under test will provide a clear and concrete API available for framework to interface with.
4) The framework will have the capability to store test results for record keeping and for further offline analysis.

To summarize proposed solution is a highly reusable Functional Test Automation Framework that can take test cases as input, provide stimulants and carry out actions on the system under test via the concrete API exposed by the SUT. The framework supports simple text based test cases that can be easily written by the person knowledgeable about system under test without having any prior software development experience. The Framework will have the capability to perform pass fail evaluations, gather and store data for further off line data analysis and statistical process control.

Framework Design

Conceptual Design: Figure-2 below shows the top level design concept of the proposed test automation framework. At the heart of the framework is a test executive, which interacts with system under test (SUT) APIs to carry out simple gets and sets on the system under test, which in turn drives the actual hardware linked with those (get / set) commands. The Framework should also be able to use, or flexible enough to adapt to industry standard APIs i.e. Web Drivers (web site testing), OPC (OLE for process control), and serial devices etc. to communicate and carry out tasks (data acquisition) on external hardware like digital / analog IO, scope and digital multi-meter etc. In addition to the above key features framework should also exhibit following desirable capabilities [2].

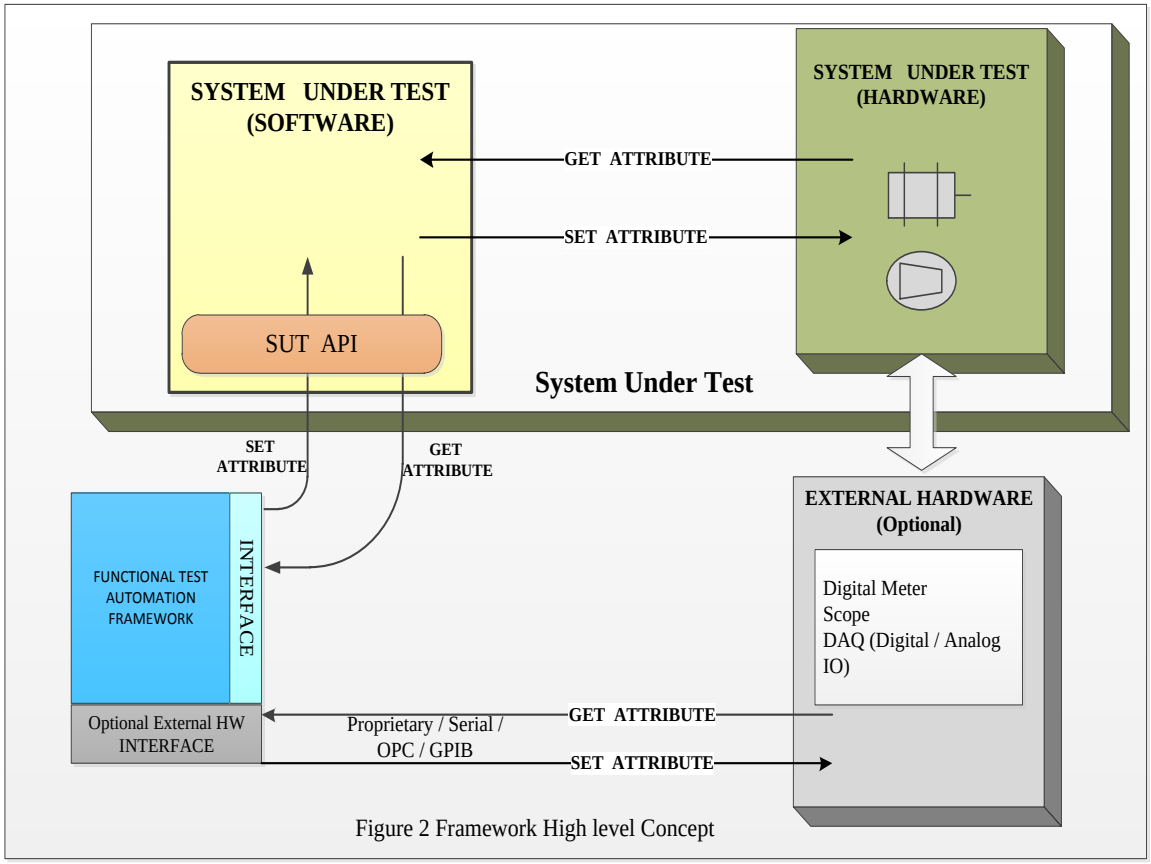


Figure 2 Framework High level Concept

- Ability to store, load and run test suite from a repository (database). This will help to maintain test suites and improve test suite reusability.
- Non programmers should be able to create tests without any software programming knowledge. Intended framework users are domain experts not software programmers.
- Mechanism to evaluate pass / fail criterion based on the specification limits provided in the test suite.
- Framework must be independent of test suites and system under test. This will provide separation of concern and it will allow to maintain test suites and framework separately.
- Ability to create summary as well as detailed report and ability to export reports into standard formats like pdf.
- Test strategy, design, and test suites must be independent of the complexities of the test automation framework.

Framework Architecture and Design Details: Figure-3, below shows the main components of the automation framework. The framework is developed in java to take advantage of Java's OS independence, thus helps increase framework's reusability on different platforms. Functional Test Automation Framework's is very database centric. MSDE (Microsoft SQL Server Data Engine) repository is used to help minimize the overhead associated with the test suite maintenance and test results storage. Test suites are completely independent of the framework however, test suites can be developed from within the framework utilizing the Scripting Utility which is one of the components of automation framework.

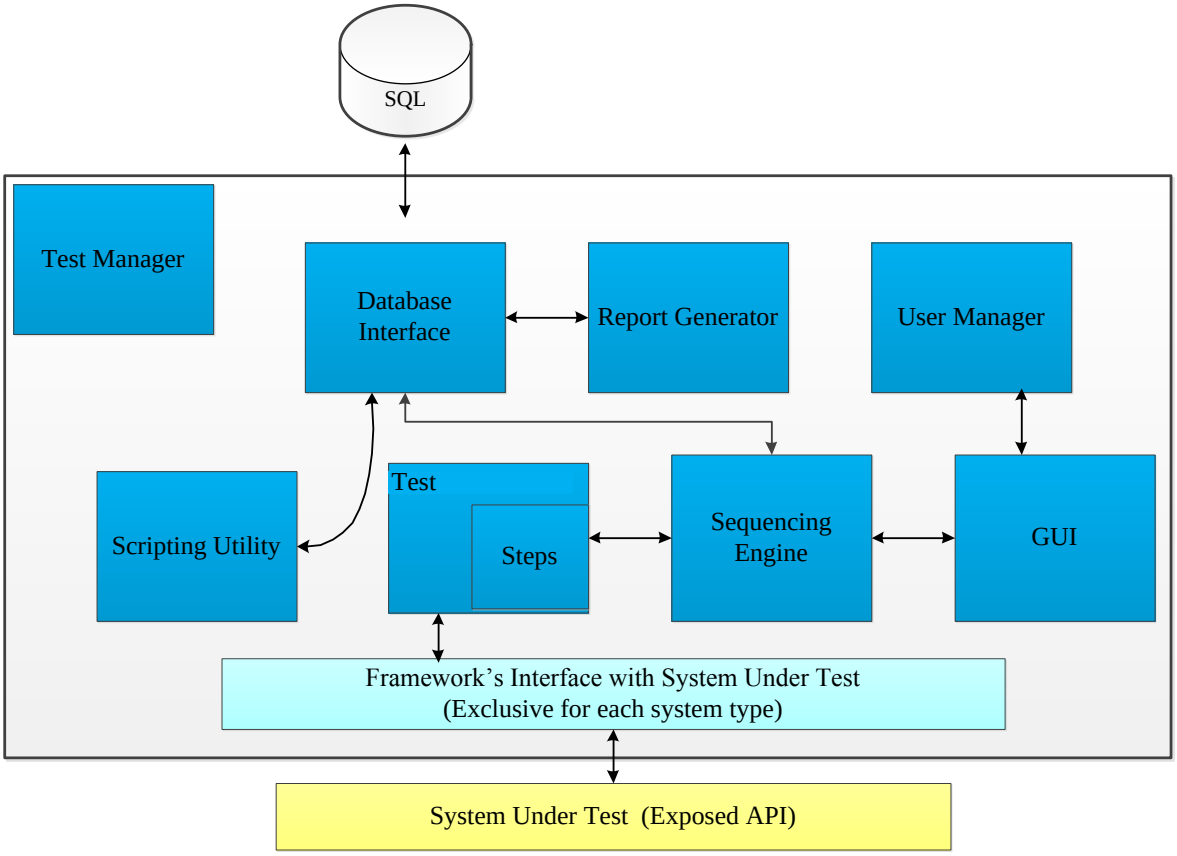


Figure 3 Test Automation Framework Architecture

Following are the details of each of the components of the automation framework.

Database interface and data repository : In order for long term maintainability of test suites and test results, Microsoft SQL server (MSDE) is used as the repository for test suites and results (Appendix-A). The database is developed using standard SQL query based interface, which will help with offline data analysis, data migration and porting the framework to any other SQL compliant database repository in future if needed. This module is used to carry out all communications with the database repository, **jtds** drivers from source forge are used for java code to communicate with SQL server database engine, which seems to perform better than **jdbc** Microsoft drivers. There are three main envisioned tasks that database interface module must perform.

- Test suites and results storage
- Test suites retrieval and loading into the framework for execution.
- Results retrieval for analysis and end user reports.

Sequencing Engine: At the heart of the framework there is a component called the sequencing engine. This module is responsible for loading and executing the test suites. Java **Test Sequence** object can have up to N **Test** objects, whereas, each test object can have up to N **Step** objects (see figure-4 entity relationship diagram below). Test sequence object and all of its contained objects (tests / steps) are instantiated by loading information from the database repository. Once a sequence is loaded into the framework, it is ready to be executed. Tests within a sequence are executed in numeric order from least to greatest, similarly **steps** within each **Test** are executed in numeric order from least to greatest. The current design is more of a simple implementation of a classic test executive however, future improvements will include conditions, jumps and looping capabilities.

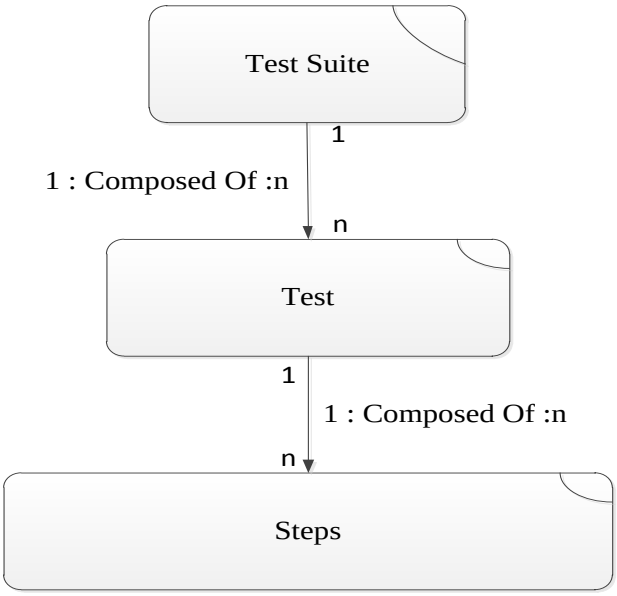


Figure-4 Test Sequence Entity Relationship

Test Executive Manager: The test manager itself is implemented as a state machine see figure-5.

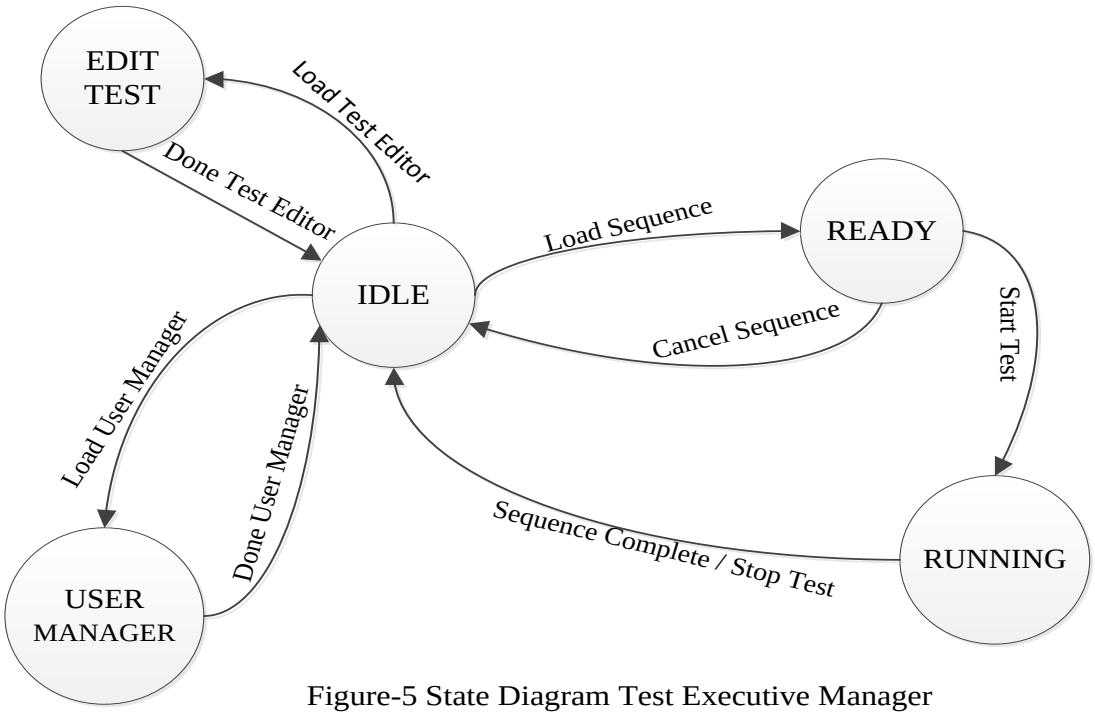


Figure-5 State Diagram Test Executive Manager

The test manager is instantiated in **IDLE** state and stays in **IDLE** state until commanded to go into one of the following states**. USER MANAGER** (used for user manager, create / delete users change privileges etc.). **EDIT TEST** (Create new / Modify existing test suite), **READY** (Test suite is loaded successfully and waiting for the user command to start execution. **RUNNING** Test sequence is executing and goes back to the **IDLE** state when test suite being executed is either completed / Sopped / or failed.

To tie the whole process together, lets look at the temporal sequence diagram (Fgure-6) which shows sequence of events involved with the loading and running of a test suite in temporal space [8].

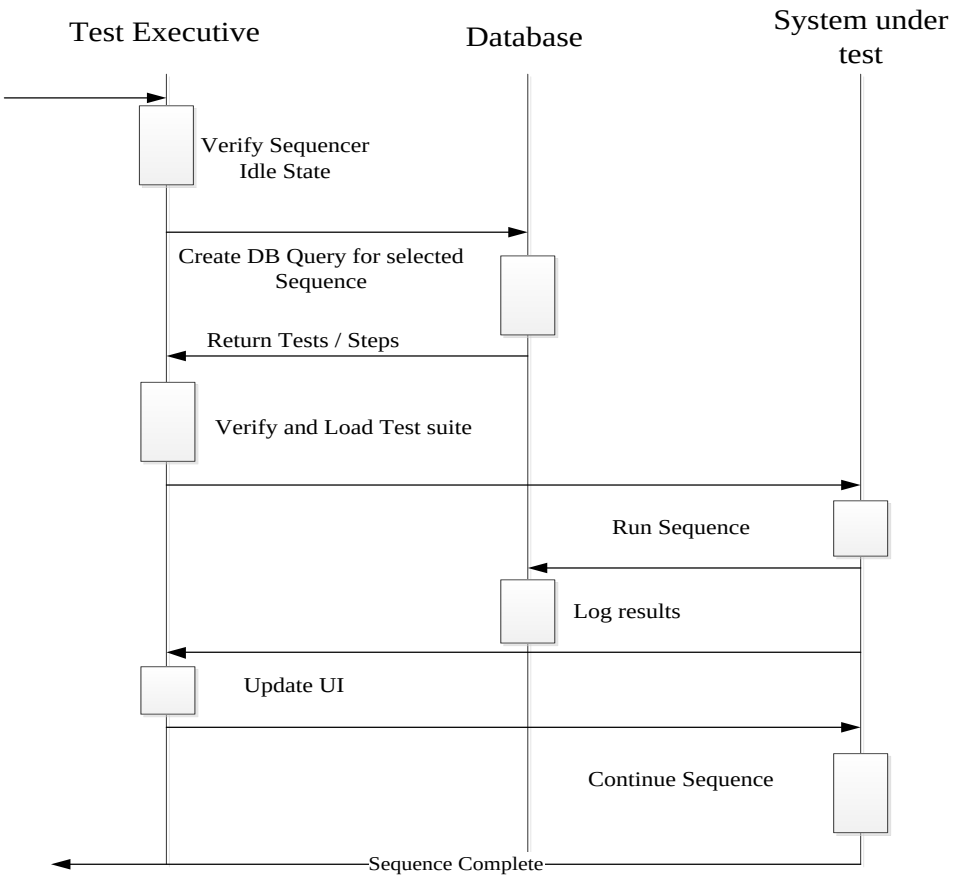


Figure-6 Temporal Sequence Diagram

Figure-7 below shows complete task decomposition involved with loading, running and logging of results for a test suite [8].

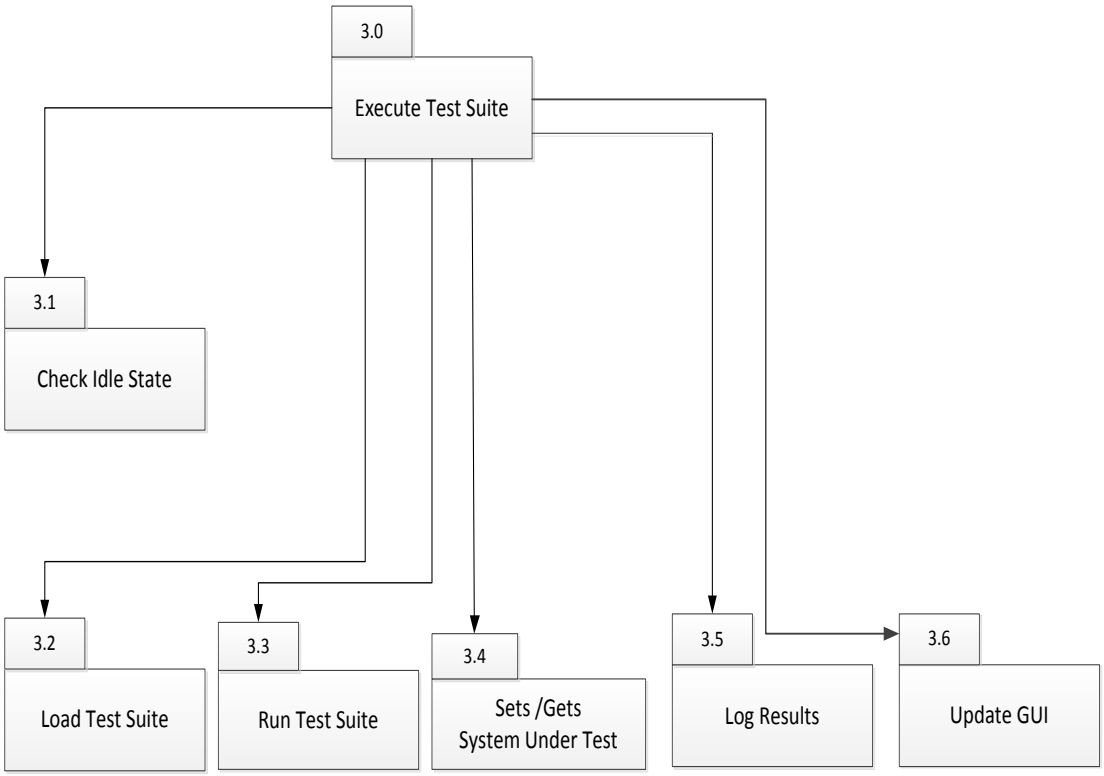


Figure-7 Task Decomposition

Scripting Utility : Since the framework is meant to be for domain experts and not the programmers, there is need for a robust and intuitive scripting utility that domain experts can use to create and edit test suites. As discussed above framework is very heavily database centric, all of the test suites and sequence are saved in the repository. The database schema has considerable amount of logic buried in it via the primary keys and sequence numbers.

The scripting utility is created where the users are given a simple form to fill out with the valid field types highlighted for each specific step type. This is accomplished via creating a table in the database and filling it out with the fields that are necessary (marked with X) and the fields that are optional (marked as O) and the fields that are irrelevant (left as null). This way when a user creates a specific step, scripting utility references the step type row from this table and highlights the fields that needs to be filled out and non essential fields are made inactive (see figure-7). More details on scripting utility are given in Appendix-A and Appendix-C.

Platform	Step Name	Sequence	FailMessage	PromptMessage	VariableValueString	PostStepDelay	PreStepDelay	VariableValue	PassFail
SERIAL	writeCrToSerialPort	X	O	X	NULL	O	O	NULL	O
SERIAL	writeLfToSerialPort	X	O	X	NULL	O	O	NULL	O
SERIAL	writeEscToSerialPort	X	O	X	NULL	O	O	NULL	O
SERIAL	writeNewPageToSerialPort	X	O	X	NULL	O	O	NULL	O
SERIAL	readDoubleFromSerialPort	X	O	X	NULL	O	O	NULL	O
SERIAL	readIntFromSerialPort	X	O	X	NULL	O	O	NULL	O
SERIAL	readStringFromSerialPort	X	O	X	X	O	O	O	O
SERIAL	writeToSerialPort	X	O	X	X	O	O	NULL	O

Table-1 Serial Interface Step types and logic

Complete Design: Details of test automation framework and key functions associated with each component are shown below in figure-8.

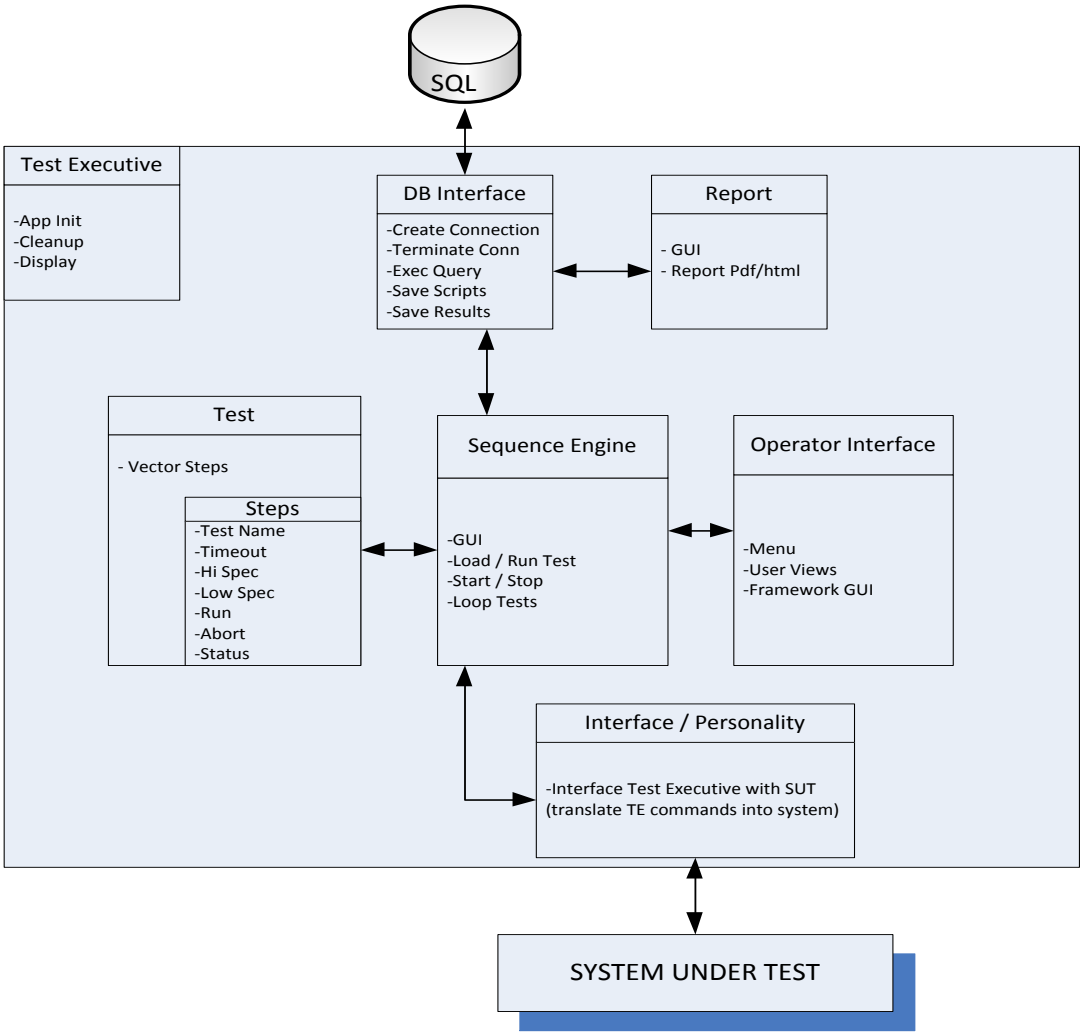


Figure-8 Framework Components Design and their salient functions

Test Suites Structure

Each Test suite consists of a three tiered architecture (figure-4). At the top of the architecture hierarchy is the test suite, a level below is all the tests that are contained within a particular test suite. The lowest level is the steps contained within each test. Steps are the actual worker functions that perform simple gets/sets on the system under test. Test executive inside the framework handles the loading and execution of test suites. Once a test suite is loaded and executed, It runs each test sequentially (according to the sequence number). Steps contained within a test are also executed sequentially. Test suites and test results are maintained independent of the framework in an SQL database, which provides nice separation of concern and easy maintenance.

Test suite is based on a very simple and intuitive text based commands (steps / worker functions), simple enough that creating test plans (scripts) won't require any programming knowledge, user will need domain expertize, and need to organize steps in numeric order, (sequence) in which these step are required to be executed. SUT will have the ability to download desired scripts locally if a local copy of SQL server exits on SUT, or map it to a remote repository via framework configuration.

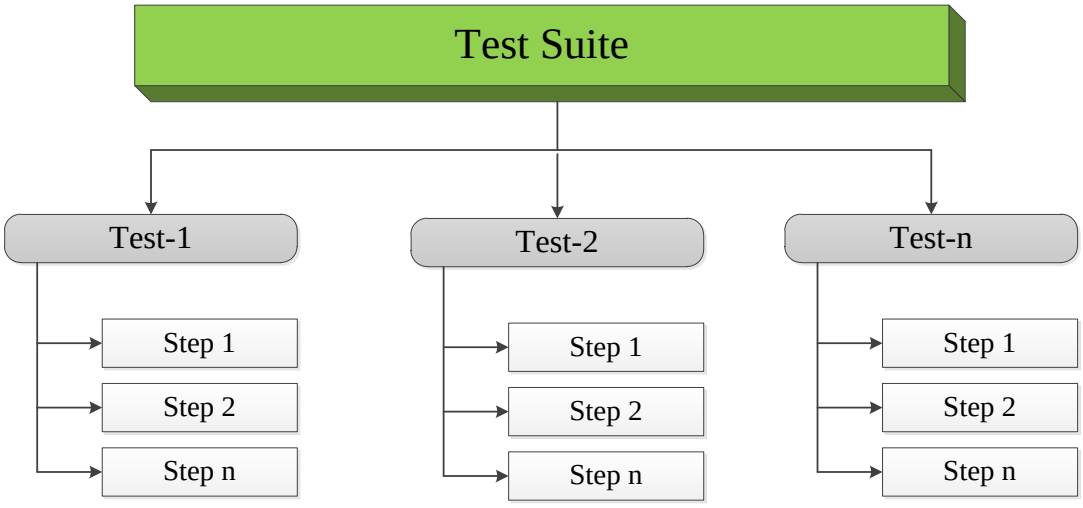


Figure-9 Anatomy of Test suite

Integration of framework and system under test

Interface between framework and system under test: This module communicates with the outside world and perform sets and gets on the system under test. The personality interface module is unique for each specific SUT (System Under Test) and has its own step dictionary (steps are concrete get / set functions). This very aspect of automation framework's design greatly enhances reusability and increases (ROI) return on investment. At the end of the day we will have collection of commonly used interfaces like Web-Drivers, CAN bus, serial, OPC, and mod bus etc. available to be used along with the proprietary interfaces for a specific interface of system under test.

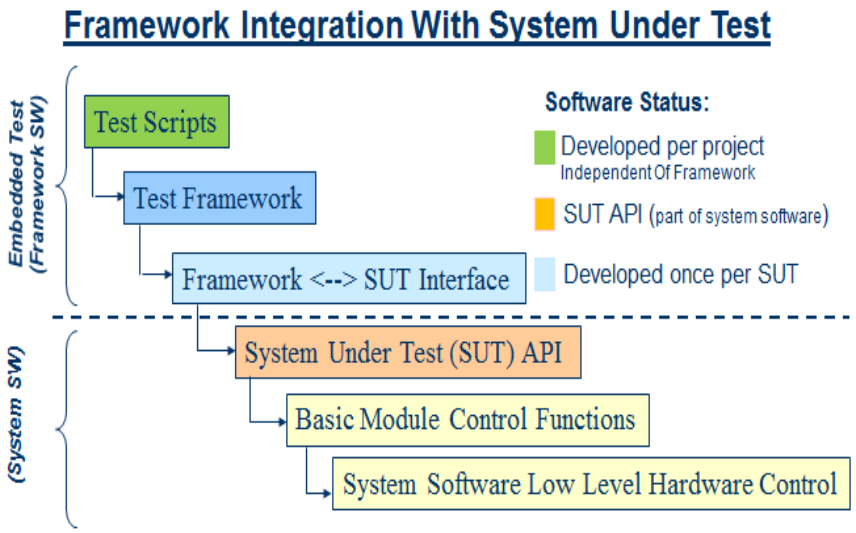


Figure-10 Integration Of Framework and SUT

Figure-9 above shows the framework to SUT integration mechanism. Framework to SUT interface layer shown in light blue color is developed only once for each system type which makes it a really good ROI considering the fact that most platforms last several years, especially industrial platforms. Even the new platforms are usually built upon their

predecessors, so what this means is creating this interface is well worth the investment. Moreover, the interface layer must be loosely coupled with SUT-API and framework itself. This will ensure framework's flexibility as well promote re-usability. Think of it as domain experts have number of these interfaces available to work with, ability to load any of these interfaces with ease, i.e. without making any changes to the code, just by a small change in the configuration file and restarting the framework. The top most layer shown above (figure-6) in green, is where domain experts deal with and develop test suites.

Evaluation & Examples

Web Application Testing: In order to evaluate test automation framework, an interface is developed for internet explorer by using Selenium2 web-drivers. Selenium provides a nice java API to be able to interface with internet explorer and other web browsers, like firefox or google chrome. In the design section above details of the interface layer are described, the interface between the test automation framework and the system under test has to be created for every unique system type. For example here we are creating an interface between our test automation framework and the IE explorer via Selenium API, however, once the interface is complete then framework can be used by domain experts (non programmers) to test any number of web application, therefore, return on investment is very high.

In a typical website testing, testers are interested in knowing that after each update to the website by the developers, that all the links on the site are intact, back/forward links work, verify timing how long to load a page, the forms that take in user input data, like login / password are working properly, and frames are displayed properly etc. Table-2 list details of interface steps (steps are concrete worker functions that carry out get /sets on the system under test) are created. Utilizing this interface a sample test suite is created which basically initializes web-driver, opens up UT blackboard website, fills in login / password fields, navigate over to one of the active courses then browse through all of the links on that course to make sure all the links are intact. In order to induce a failure, added a step where test suite looked for a linked text "Longhorn" which was not there on the page. Appendix-B lists complete detail of the test suite

WEB	initWebdriver
WEB	openPage
WEB	navigateToWebPage
WEB	findElementByNameAndFill
WEB	findTextElementByXpath
WEB	findTextElementByXpathAndFill
WEB	findElementByPartialLinkText
WEB	setImplicitWait
WEB	explicitWaitUntilElementClickable
WEB	getFrames
WEB	switchFrame
WEB	iterateOverOpenWindows
WEB	goBack
WEB	goForward
WEB	closePage
WEB	checkAlertAndAccept
WEB	getAlertAndVerifyText
WEB	checkAlertAndDismiss
WEB	closeDriver

Table-2 Web interface Step Types

Simulated Serial Device Test: In this example an interface is developed to test serial devices. Used JCOMM java libraries from SUN microsystems to create the framework to serial interface, which provided seamless integration into our automation framework, since framework is also developed in java. This means that our system under test is any serial device and we would like to communicate with this device via the test automation framework. Basically send any serial command, i.e., perform some gets / sets on the device evaluate for pass or fail criteria etc. In order to do this exercise, connected a loop back connector to the serial cable coming out of the computer. In a sense send a serial command out from port via the framework and read back the command that is just sent out. Serial interface commands are listed in table-3 and test suite is listed in appendix-B.

SERIAL	writeCrToSerialPort
SERIAL	writeLfToSerialPort
SERIAL	writeEscToSerialPort
SERIAL	writeNewPageToSerialPort
SERIAL	readDoubleFromSerialPort
SERIAL	readIntFromSerialPort
SERIAL	readStringFromSerialPort
SERIAL	writeToSerialPort

Table-3 Serial interface Step Types

Conclusion

Functional test automation framework is developed for domain experts. Domain experts are not necessarily software developers, they typically have limited to no software development skills "Programming is out of reach for many domain experts" [7]. This framework arms them with the tools, such that they can create detailed reusable test suites in terms of simple text based gets and sets from the given steps dictionary created specifically for the system under test. Framework is highly reusable and can be adapted for any new system by creating an interface layer specific for that system. Framework can be pre-populated with several industry standard protocols like CAN bus, serial, OPC (OLE for process control) etc. and other custom interfaces can be developed and loaded via the configuration file for a given system under test.

We saw that picking an off the shelf functional test framework usually seems the best choice and no brainer. However, rosy sales pitches and demos usually do not reveal how the framework will behave once it is put up against the actual system under test. At the moment of truth when all the nuances and incompatibilities between off the shelf framework and system under test become evident, it is too late, and damage has already been done in terms of lost time, product test delays and investment in terms of cost and training. The solution we have provided works well on both technical and economical fronts if following assumptions are taken into account.

For a decent ROI the system under test should have a life of at least three years or more, which usually is the case for most industrial products. API exposed by the system under test is equally important for smooth interworking between framework and SUT

If the above assumptions hold true then it is well worth the investment of time and money on an in-house, reusable test framework that can be utilized by the domain experts across the enterprise for testing needs of diverse applications. This approach is much better

economically as well as technically as oppose to employing custom test strategy for each project or acquiring something off the shelf which has hefty cost, strict license terms and periodic costly upgrades. As Carl J. Nagle emphasized “We must always remember: our ultimate goal is to simplify and perpetuate a successful automation framework” [4]

Appendix A – Database Schema

Test Suite

	Column Name	Data Type	Allow Nulls
🔑	ASSY_ID	int	☐
	Part_Number	varchar(40)	☑
	Name	varchar(60)	☐
	Description	varchar(80)	☑
	OperatorID	int	☐
	SecurityLevel	int	☐
	CurrentHistory	int	☐
	TStamp	datetime	☑
	Platform	varchar(50)	☑
	Revision	varchar(50)	☑
	EditLock	varchar(50)	☑
			☐

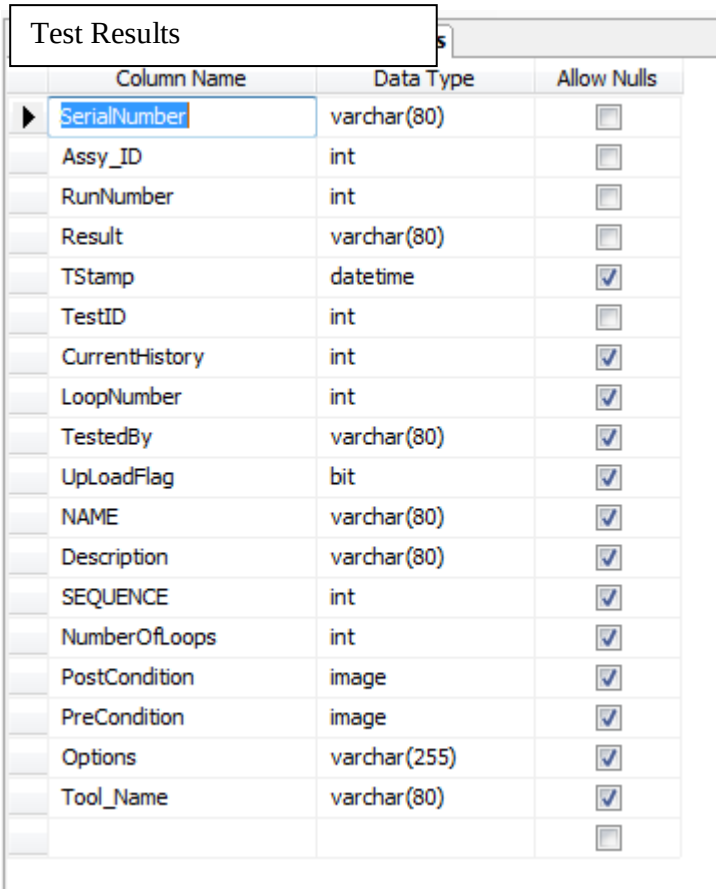
Test Suite Results

Column Name	Data Type	Allow Nulls
SerialNumber	varchar(80)	☐
Assy_ID	int	☐
RunNumber	int	☐
UserID	int	☐
Result	varchar(80)	☐
TStamp	datetime	☑
CurrentHistory	int	☑
Tool_Name	varchar(80)	☑
UpLoadFlag	bit	☑
Options	varchar(255)	☑
Revision	varchar(50)	☑
UserName	varchar(50)	☑
Part_Number	varchar(40)	☑
Name	varchar(60)	☑
Description	varchar(80)	☑
SecurityLevel	int	☑
Platform	varchar(80)	☑
TestTime	varchar(40)	☑
		☐

Test

ype	Allow Nulls
TESTS_ID	int	☐
ASSYID	int	☑
NAME	varchar(80)	☐
Description	varchar(80)	☑
OPERATOR_ID	int	☑
TSTAMP	datetime	☑
CurrentHistory	int	☑
SEQUENCE	int	☑
NumberOfLoops	int	☑
PostCondition	image	☑
PreCondition	image	☑
Options	varchar(255)	☑
		☐

Test Results



Column Name	Data Type	Allow Nulls
SerialNumber	varchar(80)	☐
Assy_ID	int	☐
RunNumber	int	☐
Result	varchar(80)	☐
TStamp	datetime	☑
TestID	int	☐
CurrentHistory	int	☑
LoopNumber	int	☑
TestedBy	varchar(80)	☑
UpLoadFlag	bit	☑
NAME	varchar(80)	☑
Description	varchar(80)	☑
SEQUENCE	int	☑
NumberOfLoops	int	☑
PostCondition	image	☑
PreCondition	image	☑
Options	varchar(255)	☑
Tool_Name	varchar(80)	☑
		☐

Step

Column Name	Data Type	Allow Nulls
STEPS_ID	int	☐
TESTS_ID	int	☐
StepType	varchar(80)	☐
Variable	varchar(512)	☑
OperatorID	int	☐
TStamp	datetime	☑
CurrentHistory	int	☑
Sequence	int	☑
FailMessage	varchar(255)	☑
PromptMessage	varchar(255)	☑
VariableValueString	varchar(512)	☑
PostStepDelay	varchar(80)	☑
PreStepDelay	varchar(80)	☑
VariableValue	varchar(80)	☑
PassFail	int	☑
Units	varchar(80)	☑
SampleRate	varchar(80)	☑
NumberOfSamples	varchar(80)	☑
DataAcquisition	int	☑
LowerLimit	varchar(80)	☑
UpperLimit	varchar(80)	☑
TimeOUT	varchar(80)	☑
MemoText	image	☑
FailAction	varchar(50)	☑
LCL	varchar(80)	☑
UCL	varchar(80)	☑
		☐

Step Results

Column Name	Data Type	Allow Nulls
SerialNumber	varchar(80)	☐
STEPS_ID	int	☐
OperatorID	int	☐
Result	varchar(80)	☐
TStamp	datetime	☑
LowerLimit	real	☑
UpperLimit	real	☑
CurrentHistory	int	☑
Units	varchar(80)	☑
AssyID	int	☑
TestsID	int	☑
RunNumber	int	☑
dataSize	int	☑
PlotData	image	☑
LoopNumber	int	☑
SampleRate	int	☑
PromptMessage	varchar(255)	☑
ActualVal	varchar(512)	☑
UpLoadFlag	bit	☑
StepType	varchar(80)	☑
Variable	varchar(512)	☑
Sequence	int	☑
FailMessage	varchar(255)	☑
VariableValueString	varchar(512)	☑
PostStepDelay	int	☑
PreStepDelay	int	☑
VariableValue	real	☑
PassFail	int	☑
NumberOfSamples	int	☑
DataAcquisition	int	☑
TimeOUT	int	☑
MemoText	image	☑
Tool_Name	varchar(80)	☑
FailAction	varchar(50)	☑
LCL	varchar(80)	☑
UCL	varchar(80)	☑
		☐

Appendix B – Sample Test Suites

Tests

Sequence	Name	NumberOfL...	Options	PreCondition	PostConditi...	Tests_
10	Web Test-1					5
20	Web Test-2					6
30	Web Test-3					7

Assv ID. 2 | Assv Rev. 1 | Availal 1B091

Assembly

[2, Web Demo, Web Demo Tests]

ADD ASSEMBLY | Opt

Serial Test 2

ADD TEST | Delete TEST

ADD STEP | EDIT STEP

Steps

Sequence	StepType	Variable	VariableValueString	PromptMessage
10	initWebdriver			Initialize web drivers
20	openPage		https://www.google.com/	open web page
25	findTextElementByXpathAndFill	q	Verification and validation UT Austin	findElementByNameAndFill
40	findTextElementByXpath	btnK		find Search button and click

Test-1 initializes web drivers and open www.google.com and searches for verification and validation.

Sequence	Name	NumberOfL...	Options	PreCondition	PostConditi...	Tests_
10	Web Test-1					5
20	Web Test-2					6
30	Web Test-3					7

2 | Assemb | [2, Web | ADD | Serial Te | ADD T | ADD

Steps

Sequence	StepType	Variable	VariableValueString
5	infoMsg		
20	openPage		https://courses.utexas.edu/
30	findTextElementByXpathAndFill	user_id	ks3528
35	findTextElementByXpathAndFill	password	*******
40	findTextElementByXpath	login	
42	getFrames		
45	switchFrame		content
50	findElementByPartialLinkText		VALIDATN
55	findElementByPartialLinkText		Announcements
65	findElementByPartialLinkText		Faculty
68	findElementByPartialLinkText		Syllabus
75	findElementByPartialLinkText		Course Documents
80	findElementByPartialLinkText		Assignments
90	findElementByPartialLinkText		My Grades
100	findElementByPartialLinkText		Communication

Test-2 Opens blackboard, logs the user in automatically and then verify different links on the

website.

Tests

Sequence	Name	NumberOfL...	Options	PreCondition	PostConditi...	Tests_
10	Web Test-1					5
20	Web Test-2					6
30	Web Test-3					7

Steps

Sequence	StepType	Variable	VariableVal...	PassFail
10	findElementByPartialLinkText	Longhorn	Longhorn	1
70	infoMsg			1
80	closePage			1
90	closeDriver			1

Test-3 performs tear down, closes the web page and then closes web driver.

Following Tests Carryout series of serial test by opening a port and then perform several writes and reads from the port. Test was simulated by looping back writes into the port.

Tests

Sequence	Name	NumberOfL...	Options	PreCondition	PostConditi...	Tests_
5	Serial Test 1					3
10	Serial Test 2					2

Assv ID. 1 Assv Rev. 1

Assembly

[1, Demo Test, Demo Test]

ADD ASSEMBLY

Serial Test 2

ADD TEST Delete TE

ADD STEP EDIT STEP

Steps

Sequence	StepType	PromptMessage	Variable	VariableVal...	VariableVal...
10	writeToSerialPort	Write String to Serial Port	PORT_1		Hello There
20	readStringFromSerialPort	Read String from Serial Port	PORT_1		Hello There
30	writeToSerialPort	Write String to Serial Port	PORT_1		100
40	readIntFromSerialPort	Read integer from Serial Port	PORT_1	100	

References

[1] Pekka Laukkanen, Data- Driven and keyword driven test automation frameworks 2006. Thesis, Helsinki University of Technology, Department Of Computer science, Finland

[2] Pekka Laukkanen, Data- Driven and keyword driven test automation frameworks 2006. Thesis, Helsinki University of Technology, Department Of Computer science, Finland

[3] Fewster and Graham, Software Test Automation, Effective use of test execution tool. Addison Wesley

[4] Carl J. Nagle, Test Automation frameworks http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm

[5] http://selenium.org

[6] Brent B. Welch, Practical Programming in Tcl and Tk, Prentice Hall

[7] Ammann & Offutt, Introduction to software testing. Cambridge

[8] Barber, K. Suzanne., 2009, Lecture Notes